🏗️ GTSuite Architecture - Technical Documentation

📋 Document Overview

Version: 1.0
Date: December 2024
Author: GTsoftware
Target Audience: Developers, System Architects, Technical Team

---

📑 Table of Contents

1. Executive Summary
2. Architectural Overview
3. Core Concepts
4. Designer Architecture (Delphi)
5. Metadata Database Schema
6. Runtime Architecture
7. Data Flow
8. Technology Stack
9. Security Model
10. Deployment Architecture

---

1. Executive Summary

1.1 What is GTSuite?

GTSuite is a **metadata-driven application development platform** that enables rapid creation of enterprise applications through a visual designer rather than traditional code-first development.

1.2 Key Characteristics

Characteristic	Description
Architecture	Metadata-Driven, Three-Tier
Designer	Delphi 11+ (Windows desktop)
Server	Node.js 18+ with Express
Client	Angular 20 / Ionic 8 (web & mobile)

Database	SQLite (design-time), MongoDB (runtime)
Multi-DB	Oracle, SQL Server, PostgreSQL, MongoDB

1.3 Core Philosophy

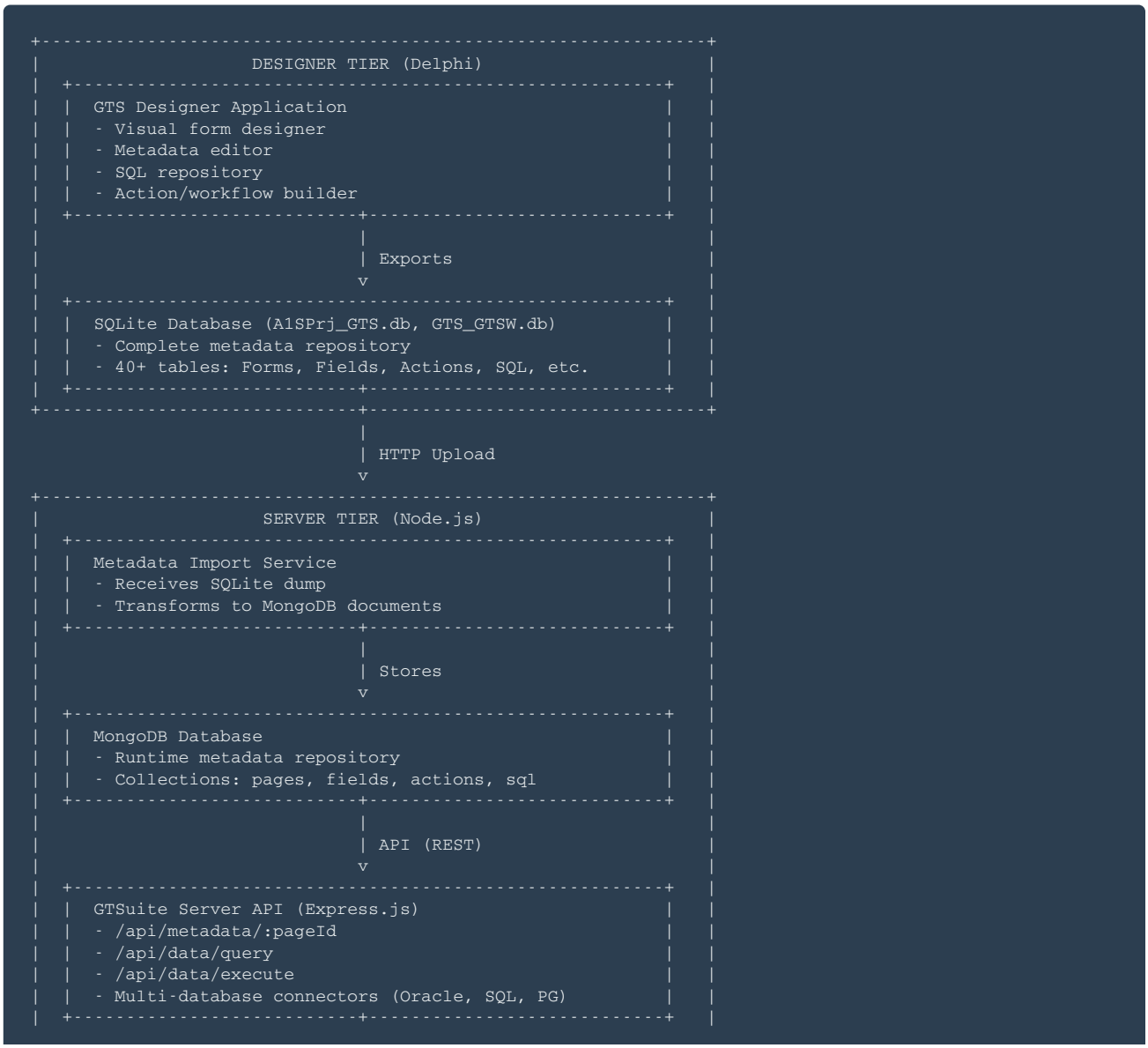
```
Design Once → Generate Metadata → Deploy Everywhere
```

- **Separation of Concerns**: Definition (metadata) ≠ Execution (runtime)
- **No Code Generation**: Runtime interprets metadata directly
- **Single Source of Truth**: Metadata in MongoDB
- **Cross-Platform**: Same metadata → Web + Mobile

---

2. Architectural Overview

2.1 Three-Tier Architecture



```
+------------------------------+-------------------------------+
                               |
                               | HTTPS / JSON
                               v
+-------------------------------------------------------------+
|                  CLIENT TIER (Angular/Ionic)                |
|  +-------------------------------------------------------+  |
|  |  GTSuite Client Application                           |  |
|  |  - Metadata renderer engine                           |  |
|  |  - Dynamic component instantiation                    |  |
|  |  - GTS Components (grid, form, toolbar, tabs)         |  |
|  +-------------------------------------------------------+  |
|                                                             |
|  Platform Targets:                                          |
|  - Web Browser (Angular)                                    |
|  - iOS App (Ionic + Capacitor)                              |
|  - Android App (Ionic + Capacitor)                          |
+-------------------------------------------------------------+
```

2.2 Metadata Lifecycle

```
1. DESIGN    → Designer creates metadata in SQLite
2. EXPORT    → SQLite database exported
3. UPLOAD    → HTTP POST to Node.js server
4. TRANSFORM → SQLite → MongoDB transformation
5. SERVE     → Metadata served via REST API
6. RENDER    → Angular client fetches & renders
7. EXECUTE   → User actions trigger API calls
8. PERSIST   → Data saved to business databases
```

---

3. Core Concepts

3.1 Metadata-Driven Development

Traditional Approach:

```
Code → Compile → Deploy → Run
```

GTSuite Approach:

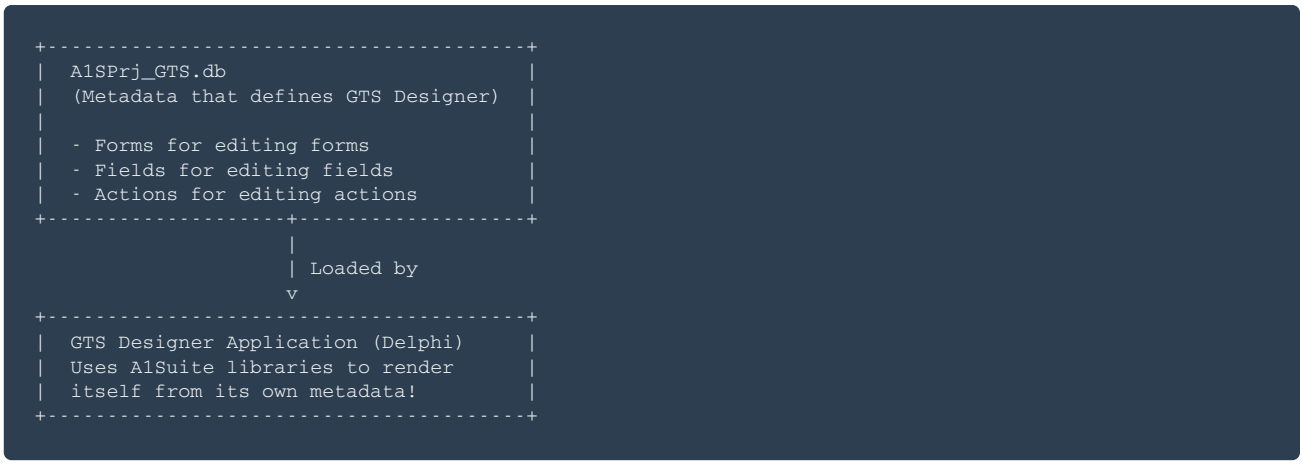
```
Metadata → Upload → Interpret → Run
```

Benefits:

- ✅ No compilation step
- ✅ Instant deployment
- ✅ Zero client-side code changes
- ✅ Centralized configuration
- ✅ Version control on metadata
- ✅ A/B testing with metadata variants

3.2 Meta-Circular Architecture

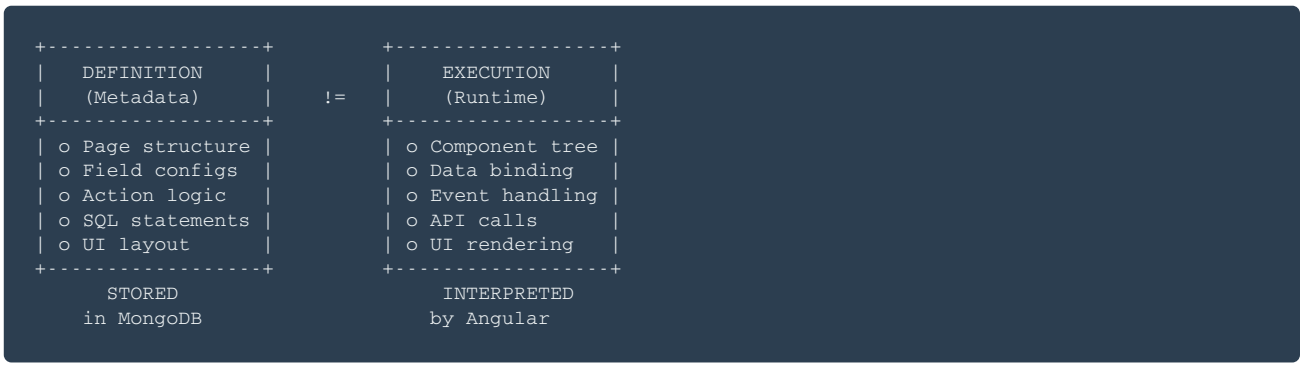
GTS Designer is built with GTSuite itself!



This means:

- The Designer eats its own dog food
- Same metadata format for Designer and user apps
- Ultimate proof of platform capability

3.3 Separation of Definition and Execution



Key Insight: The runtime NEVER stores the metadata - it fetches, caches, and interprets it on-demand.

---

4. Designer Architecture (Delphi)

4.1 Core Libraries

A1Suite Package

The foundational library containing metadata handling classes.

Key Files:

- `A1Common.pas` (73 KB, 2,077 lines)
- `A1FormDef.pas` (49 KB, 1,295 lines)
- `A1DBconn.pas` (47 KB)
- `A1UserData.pas` (28 KB)

Main Classes:

```
// TA1CommonData - Application State
TA1CommonData = class
  // Metadata Lists
  lSQLList:      TObjectList<TA1SQLDef>;
  lButtons:      TObjectList<TA1Button>;
  lConnections:  TObjectList<TA1Connections>;
  lLanguages:    TObjectList<TA1Languages>;
  lMenuItems:    TObjectList<TA1MenuItems>;

  // User & Session
  pA1User:       TA1Users;
  pA1Connection: TA1Connections;

  // Active Project
  sPrj_Id:       string;
  sRunList:      string;
end;

// TA1SQLDef - SQL Definition
TA1SQLDef = class
  iClsql_Id:           Integer;
  sClsql_Code:         string;
  sClsql_Select:       string;
  sClsql_Where:        string;
  sClsql_Orderby:      string;
  lColumns:            TObjectList<TA1Columns>;
  lSQLVariables:       TObjectList<TA1SQLVars>;
  lSQLPKColumns:       TObjectList<TA1SQLPKColumns>;
end;

// TA1Fields - Field Definition
TA1Fields = class
  iClfld_Id:           Integer;
  sClfld_Obj_Name:     string;
  sClfld_Field_Name:   string;
  sClfld_Editor_Type:  string;  // text, date, combo, lookup, etc.
  pFieldSQL:           TA1SQLDef;
  lFieldsAutoLoad:     TObjectList<TA1FieldsAutoload>;
end;

// TA1Phase - Action/Workflow Definition
TA1Phase = class
  sPhaseId:    string;
  sPhaseEvent: string;  // onClick, onLoad, onSubmit, etc.
  sObjName:    string;  // Component name
  lPhaseDet:   TObjectList<TA1PhaseDet>;
end;

// TA1PhaseDet - Action Step Details
TA1PhaseDet = class
  sPhaseId:       string;
  iOrder:         integer;
  sOperationId:   string;  // dsInsert, dsPost, setView, etc.
  pPhaseSQL:      TA1SQLDef;
  iMsgId:         integer;
  lPhaseObj:      TObjectList<TA1PhaseObj>;
end;

// TA1Grids - Grid Definition
TA1Grids = class
  iClsql_Id:           Integer;
  sClgrid_Obj_Name:    string;
  pSQLDef:             TA1SQLDef;
  pPhase:              TA1Phase;  // Grid events
end;
```

4.2 Designer Application Structure

Main Forms:

Form Module	Purpose	Lines (PAS)	Lines (DFM)
`GTS_MAINFORM`	Main MDI container	230	~4,000
`GTS_FORM_LOGIC`	**Form/page editor**	2,432	14,673

`GTS_SQL_REPOSITORY`	SQL statement manager	~800	~2,000
`GTS_METADATA_UPLOADER`	Upload to MongoDB	~400	~4,000
`GTS_MENU_DESIGNER`	Menu tree editor	~250	~500
`GTS_CONNECTIONS`	Database connections	~150	~400
`GTS_REPORTS`	Report configuration	~200	~500

Total codebase:

- 17 Pascal units (.pas)
- 16 Form definitions (.dfm)
- ~8,500 lines of code
- ~40,000 lines of DFM (visual definitions)

4.3 GTS_FORM_LOGIC - The Heart

`GTS_FORM_LOGIC.pas` is the most critical module:

Responsibilities:

1. **Page Configuration**
   - Form ID, title, URL
   - Initial action (mainInit)
2. **Dataset Management**
   - SELECT/INSERT/UPDATE/DELETE SQL
   - Data adapters
   - Master-detail relationships
3. **Field Groups & Fields**
   - Field layout (CSS Grid)
   - Editor types (text, date, combo, lookup)
   - Validations
   - Primary keys
4. **Grids**
   - Column selection
   - Column formatting
   - Events (click, doubleclick)
5. **Actions/Workflows**
   - Toolbar actions
   - Grid events
   - Form events
   - Custom code
6. **Views**

- CSS Grid layouts
- Visibility rules

7. **Tabs**
   - Tab groups
   - Tab panels

Key Methods:

```
procedure LoadPageMetadata(iPrjId: string; iFormId: integer);
procedure SavePageMetadata;
procedure LoadDataSets;
procedure LoadFields;
procedure LoadActions;
procedure LoadGrids;
procedure LoadViews;
procedure ExportToSQLite;
```

4.4 Metadata Storage (SQLite)

Two SQLite Databases:

A1SPrj_GTS.db - Designer's Own Metadata

Meta-circular! Contains the metadata that defines the GTS Designer itself.

Key Tables:

- `GtsForms` - Designer's forms
- `GtsFields` - Designer's fields
- `GtsActions` - Designer's actions
- All the same tables used for user projects

GTS_{PROJECT}.db - User Project Metadata

Example: `GTS_GTSW.db` for GTSW project

41 Core Tables:

Category	Tables	Count
Forms	GtsForms	1
Fields	GtsFields, GtsFieldsDet, GtsPageFields, GtsPageFieldsGrp, GtsFldGroups, GtsFldWizards	6
Data	GtsDataSets, GtsDataSchemas, GtsMDBCollections, GtsMDBFields, GtsMDBOperations, GtsMDBOperFields	6
SQL	GtsSQL, GtsSQLAllColumns, GtsSQLColumnsLink, GtsSQLPKColumns, GtsSQLVariables, GtsTmpSQLAllColumns	6
UI	GtsGrids, GtsGridBands, GtsColumns, GtsColumnsImg, GtsViews, GtsViewsHdr, GtsTabs, GtsTabsHdr	8
Actions	GtsActions, GtsActionsHdr, GtsExecCondRules	3
Navigation	GtsMenuTree, GtsMenuItems, GtsToolbar, GtsToolbarItems, GtsButtons	5

Reports	GtsRptReports, GtsRptGroups, GtsRptPageGroupLink, GtsRptServices	4
System	GtsConnections, GtsSetup, GtsMessages, GtsUploadConfig	4

Total: 41 tables, ~200 columns

---

5. Metadata Database Schema

5.1 Core Tables Details

GtsForms - Page Definition

```
CREATE TABLE GtsForms (
    PRJ_ID              STRING(30),
    FORM_ID             INTEGER PRIMARY KEY,
    FORM_TITLE          STRING(100),
    FORM_NAME           STRING(100),
    FORM_URL            STRING(50),        -- Angular route
    INIT_ACTION         STRING(30),        -- e.g. "mainInit"
    TXT_ID              INTEGER
);
```

Purpose: Defines a page/form in the application.

Example:

```
INSERT INTO GtsForms VALUES (
    'GTSW',           -- Project
    42,               -- Form ID
    'User Management', -- Title
    'UsersPage',      -- Component name
    'users',          -- URL: /users
    'mainInit'        -- Initial action
);
```

GtsFields - Field Configuration

```
CREATE TABLE GtsFields (
    PRJ_ID                      STRING(30),
    FORM_ID                     INTEGER,
    CLFLD_ID                    INTEGER,
    CLFLD_OBJ_NAME              STRING(100),   -- Field name
    CLFLD_FIELD_NAME            STRING(30),    -- DB column
    CLFLD_EDITOR_TYPE           STRING(30),    -- text, date, combo, lookup
    CLFLD_FLAG_ALLOW_EMPTY      STRING(1),     -- Y/N
    CLFLD_FLAG_PK               STRING(1),     -- Y/N primary key
    CLFLD_DEFAULT_VALUE         STRING(200),
    CLFLD_GRID_AREA             STRING(1000),  -- CSS Grid position
    CLFLD_OBJ_LABEL             STRING(200),   -- Label text
    CLSQL_ID                    INTEGER,       -- Lookup SQL
    CLFLDGRP_ID                 INTEGER,       -- Field group
    ACTION_NAME                 STRING(30),    -- Change event action
    BUTTON_ID                   INTEGER,       -- Attached button
    ...
);
```

Editor Types:

- `text` - Text input
- `number` - Number input
- `date` - Date picker
- `datetime` - DateTime picker
- `combo` - Dropdown (static)
- `lookup` - Dropdown (from DB)
- `textarea` - Multi-line text
- `checkbox` - Boolean
- `radio` - Radio buttons
- `password` - Password input

GtsDataSets - Data Binding

```
CREATE TABLE GtsDataSets (
    PRJ_ID                          STRING(30),
    FORM_ID                         INTEGER,
    CLDATASET_OBJ_NAME              STRING(30),     -- Dataset name
    CLSQL_ID                        INTEGER,        -- SELECT SQL
    CLDATASET_MASTER_OBJ_NAME       STRING(30),     -- Master dataset (M-D)
    CLDATASET_DATAADAPTER_OBJ_NAME  STRING(30),     -- API endpoint name
    CLDATASET_INS_SQL_ID            INTEGER,        -- INSERT SQL
    CLDATASET_UPD_SQL_ID            INTEGER,        -- UPDATE SQL
    CLDATASET_DEL_SQL_ID            INTEGER,        -- DELETE SQL
    CLDATASET_IUD_TABLE             STRING(30)      -- Target table
);
```

Master-Detail Example:

```
-- Master: Customers
INSERT INTO GtsDataSets VALUES (
    'GTSW', 42, 'CustomerDS', 100, NULL, 'customers', 101, 102, 103, 'CUSTOMERS'
);

-- Detail: Orders (linked to CustomerDS)
INSERT INTO GtsDataSets VALUES (
    'GTSW', 42, 'OrderDS', 200, 'CustomerDS', 'orders', 201, 202, 203, 'ORDERS'
);
```

GtsSQL - SQL Repository

```
CREATE TABLE GtsSQL (
    PRJ_ID              STRING(30),
    CLSQL_ID            INTEGER PRIMARY KEY,
    CLSQL_CODE          STRING(30),         -- SQL identifier
    CLSQL_TYPE          STRING(10),         -- S/I/U/D/P
    CLSQL_SELECT        STRING(10000),      -- SELECT clause
    CLSQL_FROM          STRING(5000),       -- FROM clause
    CLSQL_WHERE         STRING(5000),       -- WHERE clause
    CLSQL_ORDERBY       STRING(1000),       -- ORDER BY clause
    CLSQL_GROUPBY       STRING(1000),       -- GROUP BY clause
    CONN_CODE           STRING(20),         -- Connection
    ...
);
```

SQL Types:

- `S` - SELECT (read data)
- `I` - INSERT (create record)
- `U` - UPDATE (modify record)
- `D` - DELETE (remove record)
- `P` - PROCEDURE (stored procedure)

Example:

```
INSERT INTO GtsSQL VALUES (
    'GTSW',
    100,
    'SEL_CUSTOMERS',
    'S',
    'CUST_ID, CUST_NAME, CUST_EMAIL, CUST_PHONE',
    'CUSTOMERS',
    'CUST_ACTIVE = ''Y''',
    'CUST_NAME',
    NULL,
    'MAIN_DB'
);
```

GtsActions - Action Definitions

```
CREATE TABLE GtsActions (
    PRJ_ID              STRING(30),
    FORM_ID             INTEGER,
    ACTION_NAME         STRING(30),        -- Action identifier
    ACTION_TYPE         STRING(20),        -- See 38 action types
    ACTION_ORDER_LOGIC  INTEGER,           -- Execution order
    CUSTOM_CODE         STRING(20),        -- Custom code name
    VIEW_NAME           STRING(30),        -- Target view
    DATA_ADAPTER        STRING(30),        -- Dataset name
    CLSQL_ID            INTEGER,           -- SQL to execute
    FIELDGRP_ID         INTEGER,           -- Field group
    CLDATASET_OBJ_NAME  STRING(30),        -- Dataset
    CLMSG_ID            INTEGER,           -- Message
    EXEC_ACTION         STRING(30),        -- Action to call
    EXEC_COND_ARRAY     STRING(200),       -- Conditions
    TOOLBAR_OBJ_NAME    STRING(30),        -- Toolbar
    CLGRID_OBJ_NAME     STRING(30),        -- Grid
    ...
);
```

38 Action Types: (Already documented in GTSUITE_ACTIONS_REFERENCE.md)

Example - mainInit Action:

```
-- Step 1: Load data
INSERT INTO GtsActions VALUES (
    'GTSW', 42, 'mainInit', 'getData', 1, NULL, NULL, 'CustomerDS', NULL, ...
);

-- Step 2: Show view
INSERT INTO GtsActions VALUES (
    'GTSW', 42, 'mainInit', 'setView', 2, NULL, 'ListView', NULL, NULL, ...
);
```

GtsActionsHdr - Action Groups

```
CREATE TABLE GtsActionsHdr (
    PRJ_ID          STRING(30),
    FORM_ID         INTEGER,
    ACTION_NAME     STRING(30)      -- Groups actions with same name
);
```

Purpose: Groups multiple action steps that belong together (e.g., all steps of "mainInit" action).

GtsGrids - Grid Configuration

```
CREATE TABLE GtsGrids (
    PRJ_ID                      STRING(30),
    FORM_ID                     INTEGER,
    CLGRID_OBJ_NAME             STRING(30),     -- Grid name
    CLGRID_CSS_AREA             STRING(50),     -- CSS Grid area (e.g. "R2C1")
    CLDATASET_OBJ_NAME          STRING(30),     -- Data source
    CLGRID_FLAG_MULTISELECT     STRING(1),      -- Y/N
    CLGRID_FLAG_PAGINATION      STRING(1),      -- Y/N
    CLGRID_PAGE_SIZE            INTEGER,        -- Rows per page
    ACTION_CLICK                STRING(30),     -- Click action
    ACTION_DBLCLICK             STRING(30),     -- Double-click action
    ...
);
```

GtsViews - Layout Views

```
CREATE TABLE GtsViews (
    PRJ_ID          STRING(30),
    FORM_ID         INTEGER,
    VIEW_NAME       STRING(30),         -- View identifier
    VIEW_DESCR      STRING(100),        -- Description
    VIEW_STYLE      STRING(5000)        -- CSS Grid template
);
```

CSS Grid Example:

```
INSERT INTO GtsViews VALUES (
    'GTSW',
    42,
    'ListView',
    'List View with Grid',
    'display: grid; grid-template: "toolbar" 50px "grid" 1fr / 1fr;'
);
```

5.2 Table Relationships

```
GtsForms (1)
    |
    +-- GtsDataSets (N)
    |       |
    |       +-- GtsSQL (1)
    |               |
```

```
|               +-- GtsSQLAllColumns (N)
|               +-- GtsSQLVariables (N)
|               +-- GtsSQLPKColumns (N)
|
+-- GtsFields (N)
|       |
|       +-- GtsSQL (0..1) - Lookup
|       +-- GtsFldGroups (1)
|       +-- GtsButtons (0..1)
|
+-- GtsGrids (N)
|       |
|       +-- GtsDataSets (1)
|       +-- GtsColumns (N)
|
+-- GtsActions (N)
|       |
|       +-- GtsActionsHdr (1)
|       +-- GtsSQL (0..1)
|       +-- GtsExecCondRules (0..N)
|
+-- GtsViews (N)
|       |
|       +-- GtsViewsHdr (1)
|
+-- GtsTabs (N)
|       |
|       +-- GtsTabsHdr (1)
|
+-- GtsToolbar (N)
|       |
|       +-- GtsToolbarItems (N)
|               |
|               +-- GtsButtons (1)
|
+-- GtsMenuTree (N)
        |
        +-- GtsMenuItems (N)
```

---

6. Runtime Architecture

6.1 Node.js Server

Technology Stack:

- Node.js 18+
- Express 4.18.2
- MongoDB 6.20.0
- Database Drivers:
  - `oracledb` 6.0.3
  - `mssql` 11.0.1
  - `pg` 8.11.3
  - `better-sqlite3` 5.1.6

Core Modules:

```
src/
├── server.js                    # Entry point
├── config/
│   ├── database.js              # MongoDB connection
│   └── connections.js           # Multi-DB configs
├── routes/
│   ├── metadata.routes.js       # /api/metadata/*
│   ├── data.routes.js           # /api/data/*
│   ├── auth.routes.js           # /api/auth/*
```

```
│   └── reports.routes.js        # /api/reports/*
├── controllers/
│   ├── metadata.controller.js   # Metadata CRUD
│   ├── data.controller.js       # Data operations
│   └── auth.controller.js       # Authentication
├── services/
│   ├── metadata.service.js      # Metadata business logic
│   ├── query.service.js         # SQL generation & execution
│   └── connection.service.js    # DB connection pooling
└── models/
    ├── page.model.js            # MongoDB schemas
    ├── dataset.model.js
    └── action.model.js
```

Key API Endpoints:

```
// Metadata API
GET  /api/metadata/pages/:pageId          // Get page metadata
GET  /api/metadata/menu/:projectId        // Get menu tree
POST /api/metadata/upload                 // Upload SQLite dump

// Data API
POST /api/data/query                      // Execute SELECT
POST /api/data/execute                    // Execute INSERT/UPDATE/DELETE
POST /api/data/procedure                  // Execute stored procedure

// Auth API
POST /api/auth/login                      // User login
POST /api/auth/2fa/verify                 // 2FA verification
GET  /api/auth/profile                    // User profile

// Reports API
POST /api/reports/generate                // Generate report
GET  /api/reports/pdf/:reportId           // Download PDF
```

6.2 MongoDB Collections

Metadata Collections:

```
// pages collection
{
  _id: ObjectId,
  prjId: "GTSW",
  formId: 42,
  formTitle: "User Management",
  formUrl: "users",
  initAction: "mainInit",
  datasets: [...],
  fields: [...],
  grids: [...],
  actions: [...],
  views: [...],
  toolbar: {...},
  tabs: [...]
}

// sql collection
{
  _id: ObjectId,
  prjId: "GTSW",
  sqlId: 100,
  sqlCode: "SEL_CUSTOMERS",
  sqlType: "S",
  sqlSelect: "CUST_ID, CUST_NAME...",
  sqlFrom: "CUSTOMERS",
  sqlWhere: "CUST_ACTIVE = 'Y'",
  columns: [...],
  variables: [...],
  pkColumns: [...]
}

// connections collection
{
  _id: ObjectId,
  connCode: "MAIN_DB",
  connType: "oracle",
```

```
  connPool: "main_pool",
  connServer: "192.168.1.100",
  connPort: 1521,
  connDatabase: "PRODDB",
  connUser: "app_user",
  connPassw: "encrypted..."
}
```

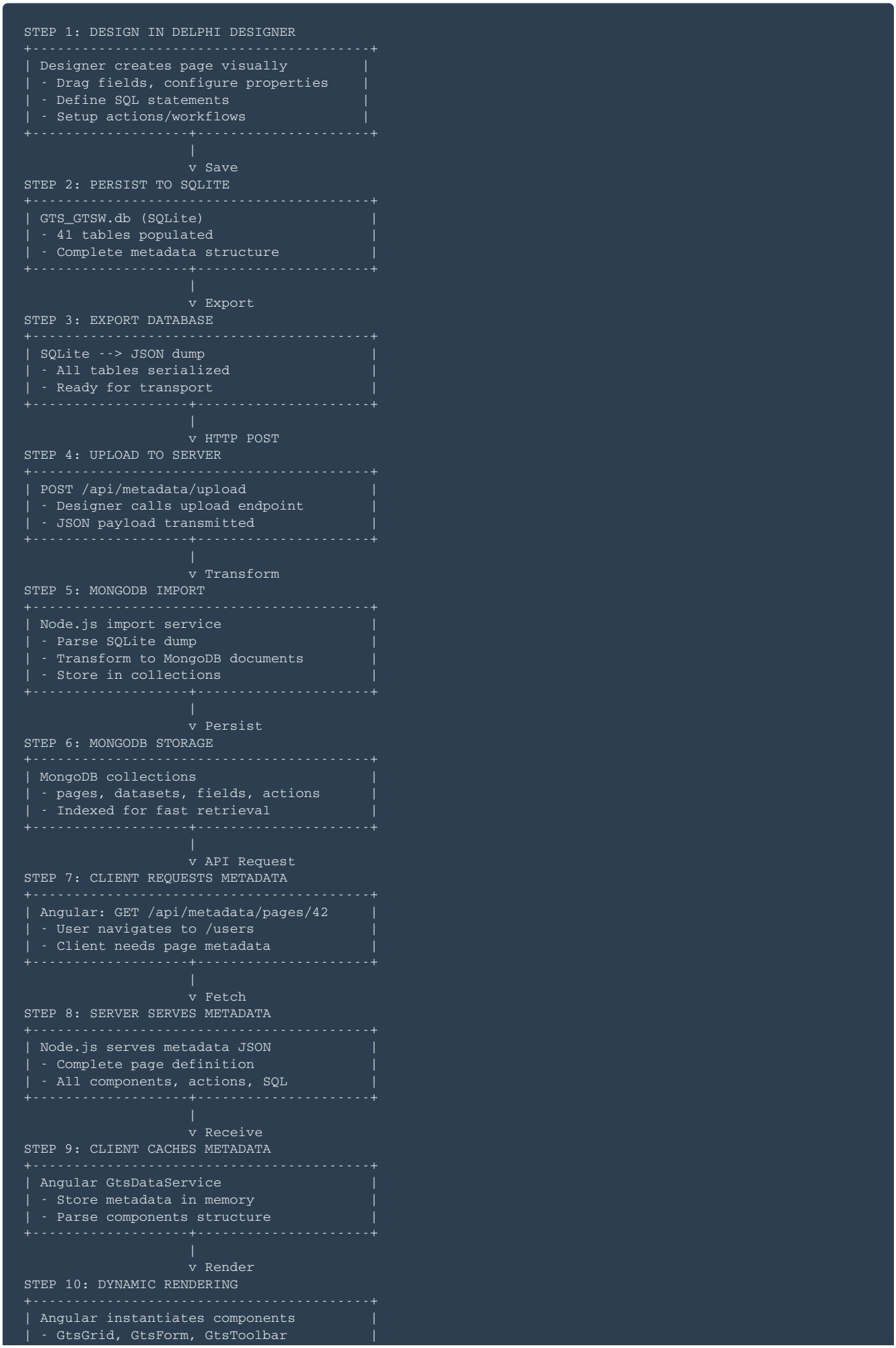
6.3 Data Flow - Query Execution

```
Client Request
    |
    | POST /api/data/query
    | { dataAdapter: "customers", params: {...} }
    |
    v
+-------------------------------------+
|  Query Controller                   |
|  - Validate request                 |
|  - Extract parameters               |
+-----------------+-------------------+
                  |
                  v
+-------------------------------------+
|  Metadata Service                   |
|  - Fetch SQL metadata from MongoDB  |
|  - Get: sqlSelect, sqlFrom, where   |
|  - Get: variables, columns          |
+-----------------+-------------------+
                  |
                  v
+-------------------------------------+
|  Query Builder                      |
|  - Substitute variables             |
|  - Build complete SQL               |
|  - Add pagination/ordering          |
+-----------------+-------------------+
                  |
                  v
+-------------------------------------+
|  Connection Pool                    |
|  - Get connection by connCode       |
|  - Select appropriate driver        |
|  - (Oracle/SQL Server/PostgreSQL)   |
+-----------------+-------------------+
                  |
                  v
+-------------------------------------+
|  Database Execution                 |
|  - Execute SQL query                |
|  - Fetch results                    |
|  - Map to JSON                      |
+-----------------+-------------------+
                  |
                  v
+-------------------------------------+
|  Response Formatter                 |
|  - Apply column metadata            |
|  - Format dates/numbers             |
|  - Add pagination info              |
+-----------------+-------------------+
                  |
                  v
    JSON Response to Client
    {
      success: true,
      data: [...],
      totalRows: 1250,
      page: 1,
      pageSize: 50
    }
```

---

7. Data Flow

7.1 Complete Metadata Journey (11 Steps)



```
| - Apply metadata properties             |
| - Bind data                             |
+-----------------+-----------------------+
                  |
                  v Display
STEP 11: USER SEES PAGE
+---------------------------------------+
| Fully functional page rendered        |
| - Grid with data                      |
| - Form fields ready                   |
| - Toolbar buttons active              |
+---------------------------------------+
```

7.2 Action Execution Flow

```
User Clicks "New" Button
    |
    | onClick event
    v
+-----------------------------+
| GtsToolbar Component        |
| - Emits toolbarSelect event |
+-------------+---------------+
              |
              | Event: {action: "toolbar_new"}
              v
+-----------------------------+
| GtsDataService              |
| - Lookup action "toolbar_new"|
| - Find action steps         |
+-------------+---------------+
              |
              | Actions: [dsInsert, clearFields, setView]
              v
+-----------------------------+
| Action Executor             |
| - Execute in sequence       |
+-------------+---------------+
              |
              +--- Step 1: dsInsert
              |       +--- Set dataset to INSERT mode
              |
              +--- Step 2: clearFields
              |       +--- Empty all form fields
              |
              +--- Step 3: setView
                      +--- Show "EditView"
```

7.3 Data Persistence Flow

```
User Clicks "Save" Button
    |
    v
+-------------------------------------+
| Step 1: saveFormData action         |
| - Read values from form fields      |
| - Update pageFields in memory       |
+-----------------+-------------------+
                  |
                  v
+-------------------------------------+
| Step 2: dsPost action               |
| - Prepare INSERT/UPDATE statement   |
| - Substitute field values           |
| - POST /api/data/execute            |
+-----------------+-------------------+
                  |
                  | HTTP Request
                  v
+-------------------------------------+
| Node.js Server                      |
| - Fetch SQL metadata                |
| - Build INSERT/UPDATE statement     |
| - Execute on target database        |
| - (Oracle/SQL Server/PostgreSQL)    |
```

---

8. Technology Stack

8.1 Designer Tier

Technology	Version	Purpose
Delphi	11+	Core language
VCL	-	UI framework
DevExpress	Latest	Advanced UI components
FireDAC	-	Database connectivity
SQLite	3.x	Metadata storage
Indy	10+	HTTP client

Third-Party Libraries:

- DevExpress VCL (cxGrid, cxTreeList, dxRibbon)
- FastReport VCL (report preview)

8.2 Server Tier

Technology	Version	Purpose
Node.js	18+	Runtime
Express	4.18.2	Web framework
MongoDB	6.20.0	Metadata storage
oracledb	6.0.3	Oracle connector
mssql	11.0.1	SQL Server connector

pg	8.11.3	PostgreSQL connector
better-sqlite3	5.1.6	SQLite connector
jsonwebtoken	9.0.2	JWT authentication
bcrypt	5.1.1	Password hashing
otplib	12.0.1	2FA/TOTP
multer	1.4.5	File upload

8.3 Client Tier

Technology	Version	Purpose
Angular	20.0.0	Framework
Ionic	8.0.0	Mobile framework
Capacitor	7.4.4	Native bridge
TypeScript	5.8.0	Language
RxJS	7.8.0	Reactive programming
DevExtreme	25.1.6	UI components

DevExtreme Components Used:

- dx-data-grid
- dx-form
- dx-toolbar
- dx-tabs
- dx-lookup
- dx-date-box
- dx-text-box

---

9. Security Model

9.1 Authentication

JWT-Based Authentication:

```
1. Login Request
   POST /api/auth/login
   { username, password }

2. Server Validates
   - Check credentials
   - Verify user active
```

```
   - Check 2FA enabled

3. 2FA Challenge (if enabled)
   - Generate OTP
   - Send via email/SMS
   - Wait for verification

4. Issue JWT Token
   - Payload: userId, role, permissions
   - Expiry: 24 hours
   - Algorithm: HS256

5. Client Stores Token
   - localStorage (web)
   - Capacitor Storage (mobile)
```

9.2 Authorization

Role-Based Access Control (RBAC):

```
User → Roles → Permissions → Granted Objects
```

Granted Objects:

- Pages (forms)
- Actions
- Menu items
- Reports

Database Tables:

- `GtsUsers` - User accounts
- `GtsRoles` - Role definitions
- `GtsUserRoles` - User-role assignments
- `GtsGrantedObj` - Object permissions
- `GtsGrantedObjRoles` - Role-permission mapping

9.3 SQL Injection Prevention

Parameterized Queries:

```
// SAFE - Uses parameters
const sql = 'SELECT * FROM CUSTOMERS WHERE CUST_ID = :id';
const params = { id: userId };
await executeQuery(sql, params);

// BLOCKED - String concatenation not allowed
// const sql = `SELECT * FROM CUSTOMERS WHERE CUST_ID = ${userId}`;
```

Variable Substitution:

- Variables defined in metadata: `@PARAM_CUST_ID`
- Runtime substitution with validation
- Type checking (string, number, date)
- SQL escaping applied automatically

9.4 Connection Security

Encrypted Passwords:

- Database passwords encrypted in MongoDB
- AES-256 encryption
- Keys stored in environment variables

IP Whitelist (Report Server):

- Only specific IPs can request reports
- Configured in `GTSRptServer.ini`

---

10. Deployment Architecture

10.1 Production Deployment

10.2 Docker Deployment

docker-compose.yml:

```
version: '3.8'
```

```yaml
services:
  # Node.js API Server
  gtsuite-server:
    image: gtsuite/server:latest
    ports:
      - "3000:3000"
    environment:
      - NODE_ENV=production
      - MONGODB_URI=mongodb://mongo:27017/gtsuite
      - JWT_SECRET=${JWT_SECRET}
    depends_on:
      - mongo
    restart: always

  # MongoDB
  mongo:
    image: mongo:6.0
    ports:
      - "27017:27017"
    volumes:
      - mongo-data:/data/db
    restart: always

  # Nginx Reverse Proxy
  nginx:
    image: nginx:alpine
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
      - ./ssl:/etc/nginx/ssl
    depends_on:
      - gtsuite-server
    restart: always

volumes:
  mongo-data:
```

10.3 Scaling Strategy

Horizontal Scaling:

```
+-----------------------------------+
|  Load Balancer                    |
+--------+--------------------------+
         |
         +--- Node.js Instance 1 (Docker container)
         +--- Node.js Instance 2 (Docker container)
         +--- Node.js Instance 3 (Docker container)
         +--- Node.js Instance N (Auto-scaled)
                  |
                  +--- Shared MongoDB cluster
```

Auto-Scaling Rules:

- CPU > 70% → Add instance
- CPU < 30% → Remove instance
- Min instances: 2
- Max instances: 10

---

11. Performance Considerations

11.1 Metadata Caching

Client-Side (Angular):

- Metadata cached in `GtsDataService`
- Cache per page (formId)
- Invalidated on version change
- Reduces API calls by ~80%

Server-Side (Node.js):

- Redis cache for metadata
- TTL: 1 hour
- Invalidated on upload
- Reduces MongoDB queries by ~90%

11.2 Database Connection Pooling

```
// Oracle connection pool
const oraclePool = {
  user: 'app_user',
  password: 'encrypted',
  connectString: 'server:1521/db',
  poolMin: 5,
  poolMax: 20,
  poolIncrement: 2,
  poolTimeout: 60
};
```

11.3 Query Optimization

Pagination:

- Page size: 50 rows (default)
- Server-side pagination
- OFFSET/LIMIT for SQL databases

Indexing:

- MongoDB indexes on: prjId, formId
- Business DB indexes on primary keys

---

12. Disaster Recovery

12.1 Backup Strategy

Metadata (MongoDB):

- Daily full backup (3 AM)
- Incremental every 6 hours
- Retention: 30 days
- Storage: AWS S3

Business Databases:

- Per client-specific backup policy
- GTSuite doesn't manage business data backups

12.2 Recovery Procedures

Scenario 1: MongoDB Failure

```
1. Promote secondary to primary (automatic)
2. Add new secondary node
3. Resync replica set
```

Scenario 2: Complete Data Loss

```
1. Restore MongoDB from latest backup
2. Restart Node.js servers
3. Clear client caches (version bump)
4. Verify functionality
```

---

13. Monitoring & Logging

13.1 Application Monitoring

Metrics Tracked:

- API response times
- Error rates
- Active connections
- Memory usage
- CPU usage

Tools:

- Prometheus (metrics collection)
- Grafana (dashboards)
- PM2 (process management)

13.2 Logging

Log Levels:

- ERROR - Critical failures
- WARN - Potential issues
- INFO - Important events
- DEBUG - Detailed diagnostic

Log Storage:

- Application logs → ELK Stack
- Access logs → Nginx logs
- MongoDB logs → Separate volume

---

14. Future Enhancements

14.1 Planned Features

1. **Visual Workflow Designer**
    - Drag-and-drop action builder
    - Visual action sequencing
    - Conditional branching
2. **AI-Assisted Development**
    - Natural language to metadata
    - Smart field suggestions
    - Auto-generate SQL
3. **Multi-Tenancy**
    - Tenant isolation
    - Shared infrastructure
    - Per-tenant customization
4. **Marketplace**
    - Pre-built templates
    - Community components
    - Plugin system

14.2 Performance Roadmap

1. **GraphQL API**
    - Replace REST with GraphQL
    - Reduce over-fetching
    - Real-time subscriptions
2. **Edge Caching**
    - CDN for static metadata
    - Geo-distributed nodes
    - Sub-50ms response times
3. **Serverless Functions**
    - AWS Lambda for actions
    - Auto-scaling

- Pay-per-execution

---

15. Conclusion

GTSuite represents a **paradigm shift** in application development:

Traditional Development:

```
Weeks of coding → Testing → Deployment → Maintenance
```

GTSuite Development:

```
Hours of design → Instant deployment → Zero maintenance
```

Key Achievements:

✅ **Metadata-driven** - Separation of definition and execution
✅ **Meta-circular** - Designer built with itself
✅ **Three-tier** - Delphi → Node.js → Angular
✅ **Multi-platform** - Web, iOS, Android from one codebase
✅ **Multi-database** - Oracle, SQL Server, PostgreSQL, MongoDB
✅ **Scalable** - Horizontal scaling, connection pooling
✅ **Secure** - JWT, RBAC, encrypted connections
✅ **Fast** - Caching, indexing, optimization

GTSuite enables:

- 10x faster development
- 90% less code
- Instant deployment
- Zero compilation
- Cross-platform by default

📞 Contact & Support

Email:
Website: www.gtsoftware.ch
Documentation: